Q1. Functions updated to accept array arguments. As an aside, requiring that roots are in roots[1]-roots[n] means that functions such as qsort() (or anything that assumes that all contents of the array are actually data and not padding) aren't guaranteed to work, depending on the value of the uninitialised roots[0] – to me this doesn't seem like a sensible tradeoff. How often is this convention followed?

Q2. Quartic solver written. Special cases denoted by comments. I also implemented a Newton-Raphson root-finder, and gave it the result of each real root as x0 to improve numerical accuracy. There may be some issues with convergence in the case of very close roots, but I have not encountered any in testing (if the quartic solver is doing a good job – which I think it is – the root should be close to its exact value, but the existence of pathological cases is possible).

In the case of polynomials with integer coefficients, Yap provides a result for guaranteed convergence to a root (<http://www.cs.nyu.edu/yap/book/berlin/>). This is not always satisfied for polynomials we might consider, so convergence is not guaranteed. A sensible approach would be to decrease the order of the polynomial each time a root is found (summary in <http://people.maths.ox.ac.uk/richardsonm/Rootfinding.pdf>)

Q3. Table of results below: (cplx means root is complex so phi is not a sensible thing to consider)

b r root 1 phi 1 root 2 phi 2 root 3 phi 3 root 4 phi 4

0.05 4 -159.7 -3.1291 -0.70164 -1.2237 -0.012551 -0.025101 0.71107 1.2362

0.1 4 -79.394 -3.1164 -0.69448 -1.214 -0.025414 -0.050817 0.71365 1.2397

0.15 4 -52.424 -3.1034 -0.68537 -1.2017 -0.03893 -0.07782 0.71493 1.2414

0.2 4 -38.788 -3.09 -0.67393 -1.186 -0.053507 -0.10691 0.71496 1.2414

0.25 4 -30.484 -3.076 -0.65963 -1.1662 -0.069675 -0.13913 0.71374 1.2398

0.3 4 -24.848 -3.0611 -0.64169 -1.141 -0.088178 -0.1759 0.71125 1.2365

0.35 4 -20.736 -3.0452 -0.61886 -1.1083 -0.11016 -0.21943 0.70743 1.2314

0.4 4 -17.576 -3.0279 -0.58904 -1.0646 -0.13756 -0.2734 0.70218 1.2244

0.45 4 -15.051 -3.0089 -0.54806 -1.0027 -0.17434 -0.34521 0.69537 1.2152

0.5 4 -12.971 -2.9877 -0.48425 -0.90194 -0.23181 -0.45557 0.6868 1.2036

0.55 2 -11.215 -2.9637 0.67622 1.1892 -0.35355 cplx 0.082859 cplx

0.6 2 -9.7018 -2.9362 0.66331 1.1713 -0.3474 cplx 0.1863 cplx

0.65 2 -8.3773 -2.904 0.64762 1.1494 -0.33902 cplx 0.26341 cplx

0.7 2 -7.2021 -2.8657 0.62864 1.1224 -0.32758 cplx 0.337 cplx

0.75 2 -6.1489 -2.8192 0.60568 1.0892 -0.31171 cplx 0.41394 cplx

0.8 2 -5.1997 -2.7616 0.57798 1.0481 -0.28913 cplx 0.49915 cplx

0.85 2 -4.3448 -2.6892 0.5448 0.99768 -0.25586 cplx 0.5975 cplx

0.9 2 -3.5849 -2.5975 0.50574 0.93646 -0.20484 cplx 0.71386 cplx

0.95 2 -2.9337 -2.4846 0.46144 0.86466 -0.12439 cplx 0.85042 cplx

The maximum value of b is found below:

b r

0.53592658043979 4

0.535926580439791 4

0.535926580439792 4

0.535926580439793 4

0.535926580439794 4

0.535926580439795 4

0.535926580439796 4

0.535926580439797 4

0.535926580439798 2

0.535926580439799 2

This gives the maximum value of b with 4 real roots as 0.535926580439797.



*Plot of distances to the ellipse at the maximal value of b with 4 real roots – two of the roots collapse onto a single repeated root on the blue line.*

Q4. Solving the quartic equation associated with the problem gives:

b r root 1 phi 1 root 2 phi 2 root 3 phi 3 root 4 phi 4

0.94 2 -2.737 -2.441 0.4251 0.8039 -0.065753 cplx 0.92474 cplx

d1 = 158422892.3713

d2 = 19999999.82634

theta 1 = 0.8369963964281 rad 47.95636098299 deg

theta 2 = 0.7334449838078 rad 42.02330207723 deg

The shorter distance 19999999.82634m is the distance of the probe from Jupiter's surface and has associated angle 0.7334 rad or 42.02330 degrees. The triangle between (rp, zp), the equator and the line perpendicular to the equator to (rp, zp) is a similar triangle to that between (rs,zs), (rp,zp) and the plane parallel to the equator so we can calculate the angle using tan(theta) = ((rp-rs)/(zp-zs)).

Similarly for the second part of the problem (my Library card # is 0246656362):

b r root 1 phi 1 root 2 phi 2 root 3 phi 3 root 4 phi 4

0.94 2 -2.5549 -2.3955 0.4691 0.87725 -0.087297 cplx 0.90925 cplx

d1 = 139423736.486

d2 = 1554048.365059

theta 1 = 0.7911626524031 rad 45.33028089107 deg

theta 2 = 0.660798153335 rad 37.86094529613 deg

Again, the shorter distance 1554048.365059m is the distance of the probe from Jupiter's surface, with associated angle 0.6608 rad of 37.8609 degrees.

MQ. Solving the quadratic equation associated with the recurrence relation for B = 1-A gives roots 1 & A-1 (=-B). Substituting this into the relation provided gives the first result. For the second part, if B = 1-A, x[n] = A\*x[n-1] + x[n-2] – A\*x[n-2], so if x[n-1] = x[n-2] then x[n] = x[n-2]. We have x[1] = x[0] so by induction this is true for all n.

On the first subtraction, a loss of significance occurs when x[0] = x[1] does not have a terminating binary representation, as we are subtracting two numbers of very similar (should be equal) size. This loss of significance is multiplied by (-B)^n, the second solution of the recurrence equation, which becomes large very rapidly as we have supplied large values of B. Precisely, the error grows as (loss of sig)\*(-B)^n, where loss of sig can be up to half of the significant digits in the value for x[0]. Therefore, float will have a larger loss of sig than double, which in turn has a smaller value than long double. (-B) is set by the relation B = -I1 at runtime.

I'm using gcc and don't yet have icc on my computer so can't test compilation to Intel C, however I would expect differences to come from implementations of floating point arithmetic and loss of significance from that. Additionally, long double may be specified differently for different hardware-compiler pairs.

According to Wikipedia, MS Visual C++ compiler uses long double as a synonym for double, so we would expect no difference in output between long double and double cases on this. Intel C++ compiler is also referenced below:[Wikipedia http://en.wikipedia.org/wiki/Long\_double]

“On the [x86 architecture](http://en.wikipedia.org/wiki/X86_architecture), most [C](http://en.wikipedia.org/wiki/C_(programming_language)) compilers implement long double as the [80-bit extended precision](http://en.wikipedia.org/wiki/Extended_precision" \l "x86_Extended_Precision_Format) type supported by x86 hardware (sometimes stored as 12 or 16 bytes to maintain [data structure alignment](http://en.wikipedia.org/wiki/Data_structure_alignment)), as specified in the [C99](http://en.wikipedia.org/wiki/C99) / [C11](http://en.wikipedia.org/wiki/C11_(C_standard_revision)) standards (IEC 60559 floating-point arithmetic (Annex F)). An exception is [Microsoft Visual C++](http://en.wikipedia.org/wiki/Microsoft_Visual_C%2B%2B) for x86, which makeslong double a synonym for double.[[2]](http://en.wikipedia.org/wiki/Long_double" \l "cite_note-2) The [Intel C++ compiler](http://en.wikipedia.org/wiki/Intel_C%2B%2B_compiler) on Microsoft Windows supports extended precision, but requires the /Qlong‑double switch for long double to correspond to the hardware's extended precision format.[[3]](http://en.wikipedia.org/wiki/Long_double" \l "cite_note-3)”

For GCC: [Same source]

“With the [GNU C Compiler](http://en.wikipedia.org/wiki/GNU_C_Compiler), long double is 80-bit extended precision on x86 processors regardless of the physical storage used for the type (which can be either 96 or 128 bits),[[4]](http://en.wikipedia.org/wiki/Long_double" \l "cite_note-4)On some other architectures, long double can be [double-double](http://en.wikipedia.org/wiki/Double-double_(arithmetic)) (e.g. on [PowerPC](http://en.wikipedia.org/wiki/PowerPC)[[5]](http://en.wikipedia.org/wiki/Long_double" \l "cite_note-5)[[6]](http://en.wikipedia.org/wiki/Long_double" \l "cite_note-6)[[7]](http://en.wikipedia.org/wiki/Long_double" \l "cite_note-7)) or 128-bit [quadruple precision](http://en.wikipedia.org/wiki/Quadruple_precision) (e.g. on [SPARC](http://en.wikipedia.org/wiki/SPARC)[[8]](http://en.wikipedia.org/wiki/Long_double" \l "cite_note-8)). As of gcc 4.3, a quadruple precision is also supported on x86, but as the nonstandard type \_\_float128 rather than long double.[[9]](http://en.wikipedia.org/wiki/Long_double" \l "cite_note-9)”

For cases 1, 4 & 7 (I2 = 1, I3 = 2):

x[0] = 0.5

x[1] = 0.5

x[2] = 0.5

x[3] = 0.5

x[4] = 0.5

x[5] = 0.5

x[6] = 0.5

x[7] = 0.5

x[8] = 0.5

x[9] = 0.5

x[10] = 0.5

x[11] = 0.5

x[12] = 0.5

x[13] = 0.5

x[14] = 0.5

x[15] = 0.5

x[16] = 0.5

x[17] = 0.5

x[18] = 0.5

x[19] = 0.5

x[20] = 0.5

x[21] = 0.5

x[22] = 0.5

x[23] = 0.5

x[24] = 0.5

x[25] = 0.5

For cases 3, 6 & 9 (I2 = 1, I3 = 4):

x[0] = 0.25

x[1] = 0.25

x[2] = 0.25

x[3] = 0.25

x[4] = 0.25

x[5] = 0.25

x[6] = 0.25

x[7] = 0.25

x[8] = 0.25

x[9] = 0.25

x[10] = 0.25

x[11] = 0.25

x[12] = 0.25

x[13] = 0.25

x[14] = 0.25

x[15] = 0.25

x[16] = 0.25

x[17] = 0.25

x[18] = 0.25

x[19] = 0.25

x[20] = 0.25

x[21] = 0.25

x[22] = 0.25

x[23] = 0.25

x[24] = 0.25

x[25] = 0.25

However for cases 2, 5 & 8 the recurrence relation is unstable. For example for I1 = 10001:

x[0] = 0.333333333333333

x[1] = 0.333333333333333

x[2] = 0.333333333333333

x[3] = 0.333333333332593

x[4] = 0.333333325928886

x[5] = 0.333259281450929

x[6] = -0.407259542592613

x[7] = -7406.33601880206

x[8] = -74074099.8573727

x[9] = -740815076006.918

x[10] = -7.40889157514852E+15

x[11] = -7.40963246430603E+19

x[12] = -7.41037342755246E+23

x[13] = -7.41111446489522E+27

x[14] = -7.41185557634171E+31

x[15] = -7.41259676189934E+35

x[16] = -7.41333802157553E+39

x[17] = -7.41407935537769E+43

x[18] = -7.41482076331323E+47

x[19] = -7.41556224538956E+51

x[20] = -7.4163038016141E+55

x[21] = -7.41704543199426E+59

x[22] = -7.41778713653746E+63

x[23] = -7.41852891525111E+67

x[24] = -7.41927076814264E+71

x[25] = -7.42001269521945E+75

Whereas for datatype float:

x[0] = 0.333333343267441

x[1] = 0.333333343267441

x[2] = 0.333251953125

x[3] = -0.480712890625

x[4] = -8140.943359375

x[5] = -81420904

x[6] = -814290501632

x[7] = -8.14371916532941e+15

x[8] = -8.14453298689957e+19

x[9] = -8.14534791606689e+23

x[10] = -8.14616246318035e+27

x[11] = -8.14697721118069e+31

x[12] = -8.14779174942031e+35

x[13] = -inf

x[14] = -nan

x[15] = -nan

x[16] = -nan

x[17] = -nan

x[18] = -nan

x[19] = -nan

x[20] = -nan

x[21] = -nan

x[22] = -nan

x[23] = -nan

x[24] = -nan

x[25] = -nan

CODE:

MAKEFILE:

all:

gcc main.c minmax.c lin\_root.c quad\_roots.c rcubic\_roots.c rquartic\_roots.c -o ex1c -lm

gcc mainQ3.c minmax.c lin\_root.c quad\_roots.c rcubic\_roots.c rquartic\_roots.c -o ex1cQ3 -lm

gcc mainQ4.c minmax.c lin\_root.c quad\_roots.c rcubic\_roots.c rquartic\_roots.c -o ex1cQ4 -lm

gcc mainMQ.c minmax.c lin\_root.c quad\_roots.c rcubic\_roots.c rquartic\_roots.c -o ex1cMQ -lm

header.h:

#include<complex.h>

int lin\_root(double [], double []);

int quad\_roots(double [], double []);

int rcubic\_roots(double [], double []);

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

int complex\_quad(complex double, complex double, complex double, complex double\*, complex double\*);

int complex\_cubic(complex double, complex double, complex double, complex double\*, complex double\*, complex double\*);

int complex\_lin\_root(complex double, complex double, complex double\*);

double calcQ(double, double);

double calcR(double, double, double);

double max(double, double);

double min(double, double);

int compare\_dbl(const void \*, const void \*);

double amax(double \*, int);

double amin(double \*, int);

double mid(double, double, double);

double absval(double);

double n\_raph\_quart(double [], double, int);

double n\_raph\_cub(double [], double, int);

double n\_raph\_quad(double [], double, int);

double n\_raph\_lin(double [], double, int);

minmax.c:

#include<stdio.h>

#include<math.h>

#include"header.h"

double max(double v1, double v2){

if (v1 < v2){

return(v2);

} else if (v1 > v2) {

return(v1);

} else {

return(v1);

}

}

double min(double v1, double v2){

if (v1 > v2){

return(v2);

} else if (v1 < v2) {

return(v1);

} else {

return(v1);

}

}

int compare\_dbl(const void \*x, const void \*y){

double \*v1 = (double \*)x;

double \*v2 = (double \*)y;

if (\*v1 < \*v2){

return(-1);

} else if (\*v1 > \*v2){

return(1);

} else {

return(0);

}

}

double amax(double\* vals, int siz){

double maxval;

int i;

maxval = vals[0];

for(i = 1; i < siz; i++){

if(vals[i] > maxval){

maxval = vals[i];

}

}

return(maxval);

}

double amin(double\* vals, int siz){

double minval;

int i;

minval = vals[0];

for(i = 1; i < siz; i++){

if(vals[i] < minval){

minval = vals[i];

}

}

return(minval);

}

double mid(double v1, double v2, double v3){

if (v1 < v2 && v2 < v3 || v3 < v2 && v2 < v1){

return(v2);

} else if (v2 < v1 && v1 < v3 || v3 < v1 && v1 < v2){

return(v1);

} else if (v1 < v3 && v3 < v2 || v2 < v3 && v3 < v1){

return(v3);

} else {

return(v1);

}

}

double absval(double v){

if (v > 0){

return(v);

} else if (v < 0){

return(-v);

} else if (v == 0){

return(0);

}

}

double n\_raph\_quart(double args[4], double init, int num\_iter){

double x;

int i;

x = init;

for(i = 0; i < num\_iter; i++){

x = x - (x\*x\*x\*x + args[3]\*x\*x\*x + args[2]\*x\*x + args[1]\*x + args[0])/(4.0\*x\*x\*x + 3.0\*args[3]\*x\*x + 2.0\*args[2]\*x + args[1]);

}

return(x);

}

double n\_raph\_cub(double args[3], double init, int num\_iter){

double x;

int i;

x = init;

for(i = 0; i < num\_iter; i++){

x = x - (x\*x\*x + args[2]\*x\*x + args[1]\*x + args[0])/(3.0\*x\*x + 2.0\*args[2]\*x + args[1]);

}

return(x);

}

double n\_raph\_quad(double args[3], double init, int num\_iter){

double x;

int i;

x = init;

for(i = 0; i < num\_iter; i++){

x = x - (args[2]\*x\*x + args[1]\*x + args[0])/(2.0\*args[2]\*x + args[1]);

}

return(x);

}

double n\_raph\_lin(double args[2], double init, int num\_iter){

double x;

int i;

x = init;

for(i = 0; i < num\_iter; i++){

x = x - (args[1]\*x + args[0])/(args[1]);

}

return(x);

}

rquartic\_roots.c:

#include<stdio.h>

#include<math.h>

#include<stdlib.h>

#include"header.h"

int rquartic\_roots(double \*a, double \*roots){

double b[3];

double c\_roots[4];

double qd\_roots[3];

double qr[4];

double p[2];

double q[2];

double inter[3];

double r;

int rVal, rVal2;

//printf("test\n");

if(a[0] == 0.0){ // special case i

//printf("special case i\n");

roots[1] = 0.0;

b[3] = a[2];

b[2] = a[1];

b[1] = a[0];

rVal = rcubic\_roots(b, c\_roots);

if(rVal != 1){

roots[2] = c\_roots[1];

roots[3] = c\_roots[2];

roots[4] = c\_roots[3];

qr[0] = n\_raph\_quart(a, roots[1],5);

qr[1] = n\_raph\_quart(a, roots[2],5);

qr[2] = n\_raph\_quart(a, roots[3],5);

qr[3] = n\_raph\_quart(a, roots[4],5);

qsort(qr, 4, sizeof(double), compare\_dbl);

roots[1] = qr[0];

roots[2] = qr[1];

roots[3] = qr[2];

roots[4] = qr[3];

return(4);

} else {

if(c\_roots[1] > 0.0){

//printf("special case i.i\n");

roots[2] = c\_roots[1];

roots[3] = c\_roots[2];

roots[4] = c\_roots[3];

roots[2] = n\_raph\_quart(a, roots[2],5);

return(2);

} else {

//printf("special case i.ii\n");

roots[1] = c\_roots[1];

roots[2] = 0.0;

roots[3] = c\_roots[2];

roots[4] = c\_roots[3];

roots[1] = n\_raph\_quart(a, roots[1],5);

return(2);

}

}

} else if (a[3] == 0.0 && a[1] == 0.0) { // special case ii

//printf("special case ii\n");

roots[1] = 0.0;

roots[2] = 0.0;

p[1] = 1.0;

p[0] = a[2];

rVal = quad\_roots(p, qd\_roots);

if(rVal == 0){

//printf("special case ii.i\n");

roots[3] = qd\_roots[1];

roots[4] = qd\_roots[2];

roots[3] = n\_raph\_quart(a, roots[3],5);

roots[4] = n\_raph\_quart(a, roots[4],5);

return(2);

} else {

//printf("special case ii.ii\n");

roots[3] = qd\_roots[1];

roots[4] = qd\_roots[2];

qr[0] = n\_raph\_quart(a, roots[1],5);

qr[1] = n\_raph\_quart(a, roots[2],5);

qr[2] = n\_raph\_quart(a, roots[3],5);

qr[3] = n\_raph\_quart(a, roots[4],5);

qsort(qr, 4, sizeof(double), compare\_dbl);

roots[1] = qr[0];

roots[2] = qr[1];

roots[3] = qr[2];

roots[4] = qr[3];

return(4);

}

} else if(a[3] == 0.0 && a[2] == 0.0 & a[1] == 0.0){ // special case iii

//printf("special case iii\n");

roots[1] = pow(abs(a[0]), 1.0/4.0);

roots[2] = -1.0\*pow(abs(a[0]), 1.0/4.0);

roots[3] = -1.0\*pow(abs(a[0]), 1.0/4.0);

roots[4] = pow(abs(a[0]), 1.0/4.0);

roots[1] = n\_raph\_quart(a, roots[1],5);

roots[2] = n\_raph\_quart(a, roots[2],5);

roots[3] = n\_raph\_quart(a, roots[3],5);

roots[4] = n\_raph\_quart(a, roots[4],5);

return(2);

}

// calculate coefficients of reduced cubic

b[2] = -1.0\*a[2];

b[1] = a[1]\*a[3]-4.0\*a[0];

b[0] = 4.0\*a[0]\*a[2] - (a[1]\*a[1] + a[0]\*a[3]\*a[3]);

rVal = rcubic\_roots(b, c\_roots);

//printf("coefficients of reduced cubic:\nb2 = %10.5g\tb1 = %10.5g\tb0 = %10.5g\n", b[2], b[1], b[0]);

//printf("c\_roots[3] = %10.8g\tc\_roots[2] = %10.8g\tc\_roots[1] = %20.18g\n", c\_roots[3], c\_roots[2], c\_roots[1]);

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

if(rVal == 3){

// choose largest root of rcubic\_roots

r = c\_roots[3]; // roots returned in size order

} else {

r = c\_roots[1];

}

//printf("largest cubic root is %20.18g\n", r);

p[1] = (a[3]/2.0) + sqrt((a[3]\*a[3]/4.0) + r - a[2]);

p[0] = (r/2.0) + sqrt((r\*r/4.0)-a[0]);

q[1] = (a[3]/2.0) - sqrt((a[3]\*a[3]/4.0) + r - a[2]);

q[0] = (r/2.0) - sqrt((r\*r/4.0)-a[0]);

//printf("p[1] = %lf\np[0] = %lf\nq[1] = %lf\nq[0] = %lf\n", p[1], p[0], q[1], q[0]);

//printf("p[1]\*q[0] + p[0]\*q[1] = %20.18g\na[1] = %20.18g\n", p[1]\*q[0] + p[0]\*q[1], a[1]);

//printf("p[1]\*q[0] + p[0]\*q[1] - a[1] = %e\n", p[1]\*q[0] + p[0]\*q[1] - a[1]);

if(abs(p[1]\*q[0] + p[0]\*q[1] - a[1]) < abs(p[1]\*p[0] + q[1]\*q[0] - a[1])){ // need to see which is better approximation rather than do equality test

// {+,+},{-,-} case

// {+,+}

//printf("{+,+},{-,-} case\n");

b[2] = 1;

b[1] = p[1];

b[0] = p[0];

//printf("calculating quad roots with b[2] = 1.0\tb[1] = %10.5g\tb[0] = %10.5g\n", b[1], b[0]);

rVal = quad\_roots(b,qd\_roots);

//printf("first root pair:\nr1 = %lf\nr2 = %lf\n", qd\_roots[1], qd\_roots[2]);

roots[1] = qd\_roots[1];

roots[2] = qd\_roots[2];

// {-,-}

b[2] = 1;

b[1] = q[1];

b[0] = q[0];

//printf("calculating quad roots with b[2] = 1.0\tb[1] = %10.5g\tb[0] = %10.5g\n", b[1], b[0]);

rVal2 = quad\_roots(b,qd\_roots);

roots[3] = qd\_roots[1];

roots[4] = qd\_roots[2];

//printf("second root pair:\nr3 = %lf\nr3 = %lf\n", qd\_roots[1], qd\_roots[2]);

if(rVal == 2 && rVal2 == 2){ // 4 real roots, just sort

qr[0] = n\_raph\_quart(a, roots[1],5);

qr[1] = n\_raph\_quart(a, roots[2],5);

qr[2] = n\_raph\_quart(a, roots[3],5);

qr[3] = n\_raph\_quart(a, roots[4],5);

qsort(qr, 4, sizeof(double), compare\_dbl);

roots[1] = qr[0];

roots[2] = qr[1];

roots[3] = qr[2];

roots[4] = qr[3];

return(4);

} else if (rVal == 0 && rVal2 != 0){ // first root is complex pair

//printf("first pair complex\n");

p[0] = roots[1];

p[1] = roots[2]; // store these values so we can put them to the end

roots[1] = min(roots[3], roots[4]);

roots[2] = max(roots[3], roots[4]);

roots[3] = p[0];

roots[4] = p[1];

roots[1] = n\_raph\_quart(a, roots[1],5);

roots[2] = n\_raph\_quart(a, roots[2],5);

return(2);

} else if (rVal != 0 && rVal2 == 0){ // second root is complex pair

//printf("second pair complex\n");

// roots are already in the right order

p[0] = max(roots[1], roots[2]);

p[1] = min(roots[1], roots[2]);

roots[1] = p[1];

roots[2] = p[0];

roots[1] = n\_raph\_quart(a, roots[1],5);

roots[2] = n\_raph\_quart(a, roots[2],5);

return(2);

} else if (rVal == 0 && rVal2 == 0){ // both quadratic equations have complex results

//printf("both pairs complex\n");

// roots are in the right order again

return(0);

} else if (rVal == 1 && rVal2 == 2 || rVal == 2 && rVal2 == 1) {

//printf("unhandled case in roots");

qr[0] = n\_raph\_quart(a, roots[1],5);

qr[1] = n\_raph\_quart(a, roots[2],5);

qr[2] = n\_raph\_quart(a, roots[3],5);

qr[3] = n\_raph\_quart(a, roots[4],5);

qsort(qr, 4, sizeof(double), compare\_dbl);

roots[1] = qr[0];

roots[2] = qr[1];

roots[3] = qr[2];

roots[4] = qr[3];

return(4);

}

} else {

// {+.-},{-,+} case

// {+,-}

//printf("{+.-},{-,+} case\n");

b[2] = 1;

b[1] = p[1];

b[0] = q[0];

rVal = quad\_roots(b,qd\_roots);

roots[1] = qd\_roots[1];

roots[2] = qd\_roots[2];

//printf("first root pair:\nr1 = %lf\nr2 = %lf\n", qd\_roots[1], qd\_roots[2]);

// {-,+}

b[2] = 1;

b[1] = q[1];

b[0] = p[0];

rVal2 = quad\_roots(b,qd\_roots);

roots[3] = qd\_roots[1];

roots[4] = qd\_roots[2];

//printf("second root pair:\nr3 = %lf\nr3 = %lf\n", qd\_roots[1], qd\_roots[2]);

if(rVal == 2 && rVal2 == 2){ // 4 real roots, just sort

qr[0] = n\_raph\_quart(a, roots[1],5);

qr[1] = n\_raph\_quart(a, roots[2],5);

qr[2] = n\_raph\_quart(a, roots[3],5);

qr[3] = n\_raph\_quart(a, roots[4],5);

qsort(qr, 4, sizeof(double), compare\_dbl);

roots[1] = qr[0];

roots[2] = qr[1];

roots[3] = qr[2];

roots[4] = qr[3];

return(4);

} else if (rVal == 0 && rVal2 != 0){ // first root is complex pair

//printf("first pair complex\n");

p[0] = roots[1];

p[1] = roots[2]; // store these values so we can put them to the end

roots[1] = min(roots[3], roots[4]);

roots[2] = max(roots[3], roots[4]);

roots[3] = p[0];

roots[4] = p[1];

roots[1] = n\_raph\_quart(a, roots[1],5);

roots[2] = n\_raph\_quart(a, roots[2],5);

return(2);

} else if (rVal != 0 && rVal2 == 0){ // second root is complex pair

//printf("second pair complex\n");

// roots are already in the right order

p[0] = max(roots[1], roots[2]);

p[1] = min(roots[1], roots[2]);

roots[1] = p[1];

roots[2] = p[0];

roots[1] = n\_raph\_quart(a, roots[1],5);

roots[2] = n\_raph\_quart(a, roots[2],5);

return(2);

} else if (rVal == 0 && rVal2 == 0){ // both quadratic equations have complex results

//printf("both pairs complex\n");

// roots are in the right order again

return(0);

} else if (rVal == 1 && rVal2 == 2 || rVal == 2 && rVal2 == 1) {

//printf("unhandled case in roots");

qr[0] = n\_raph\_quart(a, roots[1],5);

qr[1] = n\_raph\_quart(a, roots[2],5);

qr[2] = n\_raph\_quart(a, roots[3],5);

qr[3] = n\_raph\_quart(a, roots[4],5);

qsort(qr, 4, sizeof(double), compare\_dbl);

roots[1] = qr[0];

roots[2] = qr[1];

roots[3] = qr[2];

roots[4] = qr[3];

return(4);

}

}

}

rcubic\_roots.c:

#include<math.h>

#include<float.h>

#include<stdio.h>

#include"header.h"

int rcubic\_roots(double args[3], double roots[4]){

double q, r, q3\_r2;

double a2, a1, a0;

double root\_1, root\_2, root\_3;

double lr\_r[2], qr\_r[3];

double args\_b[2], args\_qr[2];

double y;

int rVal;

a2 = args[2];

a1 = args[1];

a0 = args[0];

//printf("solving cubic with args a2 = %10.5g\ta1 = %10.5g\ta0 = %10.5g\n", a2, a1, a0);

q = calcQ(a2, a1);

//printf("q: %g\n", q);

r = calcR(a2, a1, a0);

//printf("r: %g\n", r);

q3\_r2 = q\*q\*q - r\*r;

//printf("q3\_r2: %g\n", q3\_r2);

if (a2 == 0 && a1 == 0){ // special case i

//printf("case i\n");

roots[1] = pow(absval(a0), 1.0/3.0);

roots[1] = n\_raph\_cub(args, roots[1], 5);

roots[2] = (-1.0/2.0)\*pow(absval(a0), 1.0/3.0); // real part of complex root of unity

roots[3] = (sqrt(3.0)/2.0)\*pow(absval(a0), 1.0/3.0); // positive value of imaginary part

return(1);

} else if (a0 == 0){ // special case ii

//printf("case ii\n");

roots[1] = 0;

args\_qr[2] = 1.0;

args\_qr[1] = a2;

args\_qr[0] = a1;

rVal = quad\_roots(args\_qr, qr\_r);

roots[2] = qr\_r[1];

roots[3] = qr\_r[2];

roots[2] = n\_raph\_cub(args, roots[2], 5);

roots[3] = n\_raph\_cub(args, roots[3], 5);

return(rVal);

} else if (a0 == a1\*a2) { // special case iii

//printf("case iii\n");

root\_1 = -1.0\*a2;

if (a1 < 0){

root\_2 = sqrt(-1.0\*a1);

root\_3 = -1.0\*sqrt(-1.0\*a1);

roots[1] = min(min(root\_1, root\_2), min(root\_2, root\_3));

roots[2] = mid(root\_1, root\_2, root\_3);

roots[3] = max(max(root\_1, root\_2), max(root\_2, root\_3));

roots[1] = n\_raph\_cub(args, roots[1], 5);

roots[2] = n\_raph\_cub(args, roots[2], 5);

roots[3] = n\_raph\_cub(args, roots[3], 5);

return(3);

} else if (a1 > 0){

roots[1] = root\_1;

roots[1] = n\_raph\_cub(args, roots[1], 5);

roots[2] = 0; // there's no real part to the other roots

roots[3] = sqrt(a1); // only complex root exists

return(1);

}

} else if (3.0\*a1 == a2\*a2){ // special case iv

//printf("case iv\n");

roots[1] = (-1.0/3.0)\*a2;

roots[2] = (-1.0/3.0)\*a2;

roots[3] = (-1.0/3.0)\*a2;

roots[1] = n\_raph\_cub(args, roots[1], 5);

roots[2] = roots[1];

roots[3] = roots[1];

return(2);

} else if (r == 0){ // special case v

//printf("case v\n");

if (q >= 0){

roots[1] = -1.0\*sqrt(3.0\*q) - (1.0/3.0)\*a2;

roots[2] = (-1.0/3.0)\*a2;

roots[3] = sqrt(3.0\*q) - (1.0/3.0)\*a2;

roots[1] = n\_raph\_cub(args, roots[1], 5);

roots[2] = n\_raph\_cub(args, roots[2], 5);

roots[3] = n\_raph\_cub(args, roots[3], 5);

return(3);

} else if (q < 0) {

roots[1] = (-1.0/3.0)\*a2;

roots[2] = (-1.0/3.0)\*a2;

roots[3] = sqrt(-3.0\*q);

return(1);

}

} else if (q == 0){ // special case vi

//printf("case vi\n");

roots[1] = pow(absval(-2.0\*r), 1.0/3.0) - (a2/3.0);

roots[2] = pow(absval(-2.0\*r), 1.0/3.0)\*(1.0/2.0) - (a2/3.0);

roots[3] = pow(absval(-2.0\*r), 1.0/3.0)\*(sqrt(3.0)/2.0);

roots[1] = n\_raph\_cub(args, roots[1], 5);

return(1);

} else if (q\*q\*q < DBL\_MAX && r\*r < DBL\_MAX && q\*q\*q == r\*r){ // special case vii

//printf("case vii\n");

roots[1] = sqrt(q) - (a2/3.0);

roots[2] = sqrt(q) - (a2/3.0);

roots[3] = -1.0\*a2 - 2.0\*root\_1;

return(2);

} else {

//printf("general case\n");

if (q3\_r2 < 0){

//printf("2 complex roots\n");

//handle 2 complex roots

//printf("sqrt(-q3\_r2) = %g\n", sqrt(-q3\_r2));

//printf("sqrt(-q3\_r2) + absval(r) = %g\n", sqrt(-q3\_r2) + absval(r));

//printf("pow(sqrt(-q3\_r2) + absval(r), 1.0/3.0) = %g\n", pow(sqrt(-q3\_r2) + absval(r), 1.0/3.0));

roots[1] = (-absval(r)/r)\*(pow(sqrt(-q3\_r2) + absval(r), 1.0/3.0) + q/(pow(sqrt(r\*r - q\*q\*q) + absval(r), 1.0/3.0))) - a2/3.0;

//printf("First root: %g\n", roots[1]);

args\_qr[2] = 1.0;

args\_qr[1] = a2 + roots[1];

args\_qr[0] = -1.0\*a0/(roots[1]);

//printf("args\_qr[1] = %g\nargs\_qr[0] = %g\n", args\_qr[1], args\_qr[0]);

quad\_roots(args\_qr, qr\_r);

roots[2] = qr\_r[1];

roots[3] = qr\_r[2];

roots[1] = n\_raph\_cub(args, roots[1], 5);

return(1);

} else if (q3\_r2 >= 0){

//printf("3 real roots\n");

//printf("roots[1] = %g\n", roots[1]);

// handle 3 real roots

//printf("pow(q\*q\*q, 1.0/2.0) = %g\n", pow(q\*q\*q, 1.0/2.0));

//printf("acos(r/pow(q\*q\*q, 1.0/2.0)) = %g\n", acos(r/pow(q\*q\*q, 1.0/2.0)));

root\_1 = -2.0\*sqrt(q)\*cos((acos(r/pow(q\*q\*q, 1.0/2.0)))/3.0) - a2/3.0;

//printf("First root: %g\n", root\_1);

//printf("args\_qr[1] = %g\nargs\_qr[0] = %g\n", args\_qr[1], args\_qr[0]);

args\_qr[2] = 1.0;

args\_qr[1] = a2 + root\_1;

args\_qr[0] = -1.0\*a0/(root\_1);

rVal = quad\_roots(args\_qr, qr\_r);

root\_2 = qr\_r[1];

root\_3 = qr\_r[2];

//printf("root 2: %g\nroot 3: %g\n", root\_2, root\_3);

roots[1] = min(min(root\_1, root\_2), min(root\_2, root\_3));

roots[2] = mid(root\_1, root\_2, root\_3);

roots[3] = max(max(root\_1, root\_2), max(root\_2, root\_3));

roots[1] = n\_raph\_cub(args, roots[1], 5);

roots[2] = n\_raph\_cub(args, roots[2], 5);

roots[3] = n\_raph\_cub(args, roots[3], 5);

return(3);

}

}

}

double calcQ(double a2, double a1){

double qVal;

//qVal = (a2\*a2 - 3.\*a1)/9.;

qVal = (a2/3.0)\*(a2/3.0) - a1/3.0;

return(qVal);

}

double calcR(double a2, double a1, double a0){

double rVal;

//rVal = (2.\*a2\*a2\*a2 - 9.\*a1\*a2 + 27.\*a0)/54.;

rVal = a2\*((a2/3.0)\*(a2/3.0)\*(1/3.0) - (a1/6.0)) + a0/2.0;

return(rVal);

}

quad\_roots.c:

#include<stdio.h>

#include<math.h>

#include<float.h>

#include"header.h"

int quad\_roots(double args[3], double roots[3]) {

double a2 = args[2];

double a1 = args[1];

double a0 = args[0];

double lr\_args[2];

double lr\_r[2];

double q, a\_range, disc, lrg\_arg1, lrg\_arg2;

double maxval = DBL\_MAX;

//printf("solving quadratic with coefficients a2 = %10.5g\ta1 = %10.5g\ta0 = %10.5g\n", a2, a1, a0);

a\_range = max(max(a2,a1),max(a1,a0)) - min(min(a2,a1),min(a1,a0)); // dividing through reduces risk of fp overflow

// DEBUG PRINTS

//printf("max = %.5g\nmin = %.5g\n", max(max(a2,a1),max(a1,a0)), min(min(a2,a1),min(a1,a0)));

if (a\_range < DBL\_MAX && a\_range > 1.0e+100){

//printf("Dividing by a\_range = %.5g\n", a\_range);

a2 = a2/sqrt(a\_range);

a1 = a1/sqrt(a\_range);

a0 = a0/sqrt(a\_range);

}

// DEBUG PRINT

//printf("a2 = %.5g\n a1 = %.5g\n a0 = %.5g\n", a2, a1, a0);

if (a1 > sqrt(maxval) && (a2 > maxval/a0 || a0 > maxval/a2)) {

// divide through by one of the arguments (largest?) and go again

//printf("All arguments too large, dividing\n");

lrg\_arg1 = fmax(a2, a1);

lrg\_arg2 = fmax(lrg\_arg1, a0);

a2 = a2/lrg\_arg2;

a1 = a1/lrg\_arg2;

a0 = a0/lrg\_arg2;

disc = a1\*a1 - 4.0\*a2\*a0;

} else if (a2 == 0){ // linear equation case

lr\_args[1] = args[1];

lr\_args[0] = args[0];

lin\_root(lr\_args, lr\_r);

roots[1] = lr\_r[1];

roots[2] = lr\_r[1];

return(-1);

} else if (a0 == 0) { // one root is zero

lin\_root(lr\_args, lr\_r);

roots[1] = lr\_r[1];

roots[2] = 0.0;

return(2);

} else if (a1 == 0 && a0 < 0) { // x^2 = -c

roots[1] = sqrt(-a0)/sqrt(a2);

roots[2] = -sqrt(-a0)/sqrt(a2);

return(2);

} else {

disc = a1\*a1 - 4.0\*a2\*a0;

// DEBUG PRINT

//printf("disc = %.5g\n", disc);

if (disc < 0){

roots[1] = -a1/(2.0\*a2);

roots[2] = sqrt(-1.0\*disc)/(2.0\*a2);

return(0);

} else if (disc > 0){

if (a1 > 0){

q = -0.5\*(a1 + sqrt(disc));

} else {

q = -0.5\*(a1 - sqrt(disc));

}

// DEBUG PRINT

//printf("q = %.5g\n", q);

roots[1] = a0/q;

roots[2] = q/a2;

return(2);

} else if (disc == 0){

roots[1] = -a1/(2.0\*a2);

roots[2] = -a1/(2.0\*a2);

return(1);

}

}

}

lin\_root.c:

#include<stdio.h>

#include<math.h>

#include"header.h"

int lin\_root(double args[2], double r1[2]){

double A = args[1];

double B = args[0];

if (A == 0 && B == 0){

return(-1); // any x will satisfy this

} else if (A == 0 && B != 0){

return(0); // no x will satisfy this

} else if (A != 0 && B != 0){

r1[1] = -B/A;

r1[1] = n\_raph\_lin(args, r1[0], 5); //N-R sharpening of root

return(1);

}

}

main.c:

#include<stdio.h>

#include<math.h>

#include<complex.h>

#include"header.h"

int main(void){

double args[4];

double r[5];

double b, Y, X;

double closest;

int rVal;

printf("Name: <Tom McGrath>\n");

printf("CID: <00898098>, LIBRARY NO: <0246656362>\n");

printf("Email Address: <t.mcgrath13@imperial.ac.uk>\n");

printf("Course Code: <M5SC>\n");

printf("Time: %s\n",\_\_TIME\_\_);

printf("Date: %s\n",\_\_DATE\_\_);

X = 1.0/3.0;

Y = 1.0/3.0;

printf("%5s\t%4s\t%10s\t%10s\t%10s\t%10s\t%10s\t%10s\t%10s\t%10s\n", "b", "r", "root 1", "phi 1", "root 2", "phi 2", "root 3", "phi 3", "root 4", "phi 4");

for(b = 0.05; b < 1.0; b += 0.05){

args[3] = (2.0/b) + (2.0/(b\*Y)) - (2.0\*b)/Y;

args[2] = 0;

args[1] = (2.0/b) - (2.0/(b\*Y)) + (2.0\*b)/Y;

args[0] = -1.0;

rVal = rquartic\_roots(args, r);

if(rVal == 4){

printf("%5.2g\t%4d\t%10.5g\t%10.5g\t%10.5g\t%10.5g\t%10.5g\t%10.5g\t%10.5g\t%10.5g\n", b, rVal, r[1], 2.0\*atan(r[1]), r[2], 2.0\*atan(r[2]), r[3], 2.0\*atan(r[3]), r[4], 2.0\*atan(r[4]));

} else {

printf("%5.2g\t%4d\t%10.5g\t%10.5g\t%10.5g\t%10.5g\t%10.5g\t%10.5s\t%10.5g\t%10.5s\n", b, rVal, r[1], 2.0\*atan(r[1]), r[2], 2.0\*atan(r[2]), r[3], "cplx", r[4], "cplx");

}

}

}

mainQ3.c:

#include<stdio.h>

#include<math.h>

#include<complex.h>

#include"header.h"

int main(void){

double args[4];

double r[5];

double roots\_X[4];

double roots\_Y[4];

double b, Y, X;

double closest;

int rVal;

printf("Name: <Tom McGrath>\n");

printf("CID: <00898098>, LIBRARY NO: <0246656362>\n");

printf("Email Address: <t.mcgrath13@imperial.ac.uk>\n");

printf("Course Code: <M5SC>\n");

printf("Time: %s\n",\_\_TIME\_\_);

printf("Date: %s\n",\_\_DATE\_\_);

X = 1.0/3.0;

Y = 1.0/3.0;

printf("%20s\t%2s\n", "b", "r");

for(b = 0.535926580439790; b < 0.535926580439800; b += 0.000000000000001){

args[3] = (2.0/b) + (2.0/(b\*Y)) - (2.0\*b)/Y;

args[2] = 0;

args[1] = (2.0/b) - (2.0/(b\*Y)) + (2.0\*b)/Y;

args[0] = -1.0;

rVal = rquartic\_roots(args, r);

if(rVal == 4){

printf("%20.15g\t%2d\n", b, rVal);

} else {

printf("%20.15g\t%2d\n", b, rVal);

}

}

// Second part of question

printf("b = 0.535926580439797\n");

b = 0.535926580439797;

args[3] = (2.0/b) + (2.0/(b\*Y)) - (2.0\*b)/Y;

args[2] = 0;

args[1] = (2.0/b) - (2.0/(b\*Y)) + (2.0\*b)/Y;

args[0] = -1.0;

rVal = rquartic\_roots(args, r);

printf("%5s\t%2s\t%10s\t%10s\t%10s\t%10s\t%10s\t%10s\t%10s\t%10s\n", "b", "r", "root 1", "phi 1", "root 2", "phi 2", "root 3", "phi 3", "root 4", "phi 4");

printf("%5.2g\t%2d\t%10.5g\t%10.5g\t%10.5g\t%10.5g\t%10.5g\t%10.5g\t%10.5g\t%10.5g\n", b, rVal, r[1], 2.0\*atan(r[1]), r[2], 2.0\*atan(r[2]), r[3], 2.0\*atan(r[3]), r[4], 2.0\*atan(r[4]));

roots\_X[0] = cos(2.0\*atan(r[1]));

roots\_X[1] = cos(2.0\*atan(r[2]));

roots\_X[2] = cos(2.0\*atan(r[3]));

roots\_X[3] = cos(2.0\*atan(r[4]));

roots\_Y[0] = b\*sin(2.0\*atan(r[1]));

roots\_Y[1] = b\*sin(2.0\*atan(r[2]));

roots\_Y[2] = b\*sin(2.0\*atan(r[3]));

roots\_Y[3] = b\*sin(2.0\*atan(r[4]));

printf("X[0] = %15.10g\tY[0] = %15.10g\n", roots\_X[0], roots\_Y[0]);

printf("X[1] = %15.10g\tY[1] = %15.10g\n", roots\_X[1], roots\_Y[1]);

printf("X[2] = %15.10g\tY[2] = %15.10g\n", roots\_X[2], roots\_Y[2]);

printf("X[3] = %15.10g\tY[3] = %15.10g\n", roots\_X[3], roots\_Y[3]);

}

mainQ4.c:

#include<stdio.h>

#include<math.h>

#include<complex.h>

#include"header.h"

int main(void){

double args[4];

double r[4];

double b, Y, X, x, y, d1, d2, dX, dY, theta1, theta2;

int rVal;

printf("Name: <Tom McGrath>\n");

printf("CID: <00898098>, LIBRARY NO: <0246656362>\n");

printf("Email Address: <t.mcgrath13@imperial.ac.uk>\n");

printf("Course Code: <M5SC>\n");

printf("Time: %s\n",\_\_TIME\_\_);

printf("Date: %s\n",\_\_DATE\_\_);

// First part of question

X = 62997033.307818345/71492000.0;

Y = 62997033.307818345/71492000.0;

b = 66854000.0/71492000.0;

//printf("X=%lf\tY=%lf\n", X, Y);

printf("%5s\t%2s\t%10s\t%10s\t%10s\t%10s\t%10s\t%10s\t%10s\t%10s\n", "b", "r", "root 1", "phi 1", "root 2", "phi 2", "root 3", "phi 3", "root 4", "phi 4");

args[3] = (2.0/b) + (2.0/(b\*Y)) - (2.0\*b)/Y;

args[2] = 0;

args[1] = (2.0/b) - (2.0/(b\*Y)) + (2.0\*b)/Y;

args[0] = -1.0;

rVal = rquartic\_roots(args, r);

if(rVal == 4){

printf("%5.2g\t%2d\t%10.5g\t%10.5g\t%10.5g\t%10.5g\t%10.5g\t%10.5g\t%10.5g\t%10.5g\n", b, rVal, r[1], 2.0\*atan(r[1]), r[2], 2.0\*atan(r[2]), r[3], 2.0\*atan(r[3]), r[4], 2.0\*atan(r[4]));

} else {

printf("%5.2g\t%2d\t%10.5g\t%10.5g\t%10.5g\t%10.5g\t%10.5g\t%10.5s\t%10.5g\t%10.5s\n", b, rVal, r[1], 2.0\*atan(r[1]), r[2], 2.0\*atan(r[2]), r[3], "cplx", r[4], "cplx");

}

dX = X-cos(2.0\*atan(r[1]));

dY = Y-b\*sin(2.0\*atan(r[1]));

//printf("d1\_X = %20.16g\td1\_Y = %20.16g\n", dX, dY);

d1 = 71492000.0\*sqrt(dX\*dX + dY\*dY);

theta1 = atan(dX/dY);

dX = X-cos(2.0\*atan(r[2]));

dY = Y-b\*sin(2.0\*atan(r[2]));

d2 = 71492000.0\*sqrt(dX\*dX + dY\*dY);

theta2 = atan(dX/dY);

//printf("d2\_X = %20.16g\td2\_Y = %20.16g\n", dX, dY);

printf("d1 = %20.13g\nd2 = %20.13g\n", d1, d2);

printf("theta 1 = %20.13g rad\t%20.13g deg\ntheta 2 = %20.13g rad\t%20.13g deg\n", theta1, theta1\*180.0/M\_PI, theta2, theta2\*180.0/M\_PI);

// Second part of question

X = 46656362.0/71492000.0;

Y = 52636566.4/71492000.0;

//printf("X=%lf\tY=%lf\n", X, Y);

printf("%5s\t%2s\t%10s\t%10s\t%10s\t%10s\t%10s\t%10s\t%10s\t%10s\n", "b", "r", "root 1", "phi 1", "root 2", "phi 2", "root 3", "phi 3", "root 4", "phi 4");

args[3] = (2.0\*X/(b\*Y)) + (2.0/(b\*Y)) - (2.0\*b)/Y;

args[2] = 0;

args[1] = (2.0\*X/(b\*Y)) - (2.0/(b\*Y)) + (2.0\*b)/Y;

args[0] = -1.0;

rVal = rquartic\_roots(args, r);

if(rVal == 4){

printf("%5.2g\t%2d\t%10.5g\t%10.5g\t%10.5g\t%10.5g\t%10.5g\t%10.5g\t%10.5g\t%10.5g\n", b, rVal, r[1], 2.0\*atan(r[1]), r[2], 2.0\*atan(r[2]), r[3], 2.0\*atan(r[3]), r[4], 2.0\*atan(r[4]));

} else {

printf("%5.2g\t%2d\t%10.5g\t%10.5g\t%10.5g\t%10.5g\t%10.5g\t%10.5s\t%10.5g\t%10.5s\n", b, rVal, r[1], 2.0\*atan(r[1]), r[2], 2.0\*atan(r[2]), r[3], "cplx", r[4], "cplx");

}

//d1 = 71492000.0\*sqrt( (X-cos(2.0\*atan(r[0])))\*(X-cos(2.0\*atan(r[0]))) + (Y-b\*sin(2.0\*atan(r[0])))\*(Y-b\*sin(2.0\*atan(r[0]))) );

//d2 = 71492000.0\*sqrt( (X-cos(2.0\*atan(r[1])))\*(X-cos(2.0\*atan(r[1]))) + (Y-b\*sin(2.0\*atan(r[1])))\*(Y-b\*sin(2.0\*atan(r[1]))) );

dX = X-cos(2.0\*atan(r[1]));

dY = Y-b\*sin(2.0\*atan(r[1]));

theta1 = atan(dX/dY);

//printf("d1\_X = %20.16g\td1\_Y = %20.16g\n", dX, dY);

d1 = 71492000.0\*sqrt(dX\*dX + dY\*dY);

dX = X-cos(2.0\*atan(r[2]));

dY = Y-b\*sin(2.0\*atan(r[2]));

theta2 = atan(dX/dY);

d2 = 71492000.0\*sqrt(dX\*dX + dY\*dY);

//printf("d2\_X = %20.16g\td2\_Y = %20.16g\n", dX, dY);

printf("d1 = %20.13g\nd2 = %20.13g\n", d1, d2);

printf("theta 1 = %20.13g rad\t%20.13g deg\ntheta 2 = %20.13g rad\t%20.13g deg\n", theta1, theta1\*180.0/M\_PI, theta2, theta2\*180.0/M\_PI);

}

mainMQ.c:

#include<stdio.h>

#include<stdlib.h>

#include"header.h"

#define double long double

int main(void){

int n\_recurr, i;

int I1, I2, I3;

double A, B;

double \*x;

printf("Name: <Tom McGrath>\n");

printf("CID: <00898098>, LIBRARY NO: <0246656362>\n");

printf("Email Address: <t.mcgrath13@imperial.ac.uk>\n");

printf("Course Code: <M5SC>\n");

printf("Time: %s\n",\_\_TIME\_\_);

printf("Date: %s\n",\_\_DATE\_\_);

n\_recurr = 25; // set in question

printf("Enter integers I1, I2, I3:\n");

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

A = (double)I1 + 1.0;

B = -1.0\*(double)I1;

x = (double \*)malloc((n\_recurr+1)\*sizeof(double));

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

x[1] = x[0];

printf("x[0] = %20.15LG\nx[1] = %20.15LG\n", x[0], x[1]);

for(i = 2; i <= n\_recurr; i++){

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

printf("x[%d] = %20.15LG\n", i, x[i]);

}

free(x);

}