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.

[FIGURE GOES HERE]

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 = 160749444.6759

d2 = 22205893.42393

The shorter distance of 22205893.42393m is the distance of the probe from Jupiter's surface.

Likewise for the second part of the problem (my Library number is: 0246656362)

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

* 1. 2 -2.5549 -2.3955 0.4691 0.87725 -0.087297 cplx 0.90925 cplx

d1 = 153016704.0692

d2 = 19340426.55677

Again, the shorter distance 19340426.55677m is the distance of the probe from Jupiter's surface.

[CALC LONG AND LAT]

MQ.

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

This result is using long double for x[i], the instability grows larger as the accuracy of the datatype decreases to double and then float. 1/3 does not have a terminating decimal representation, so cannot be stored with full accuracy in floating point form. B is negative, so each recurrence is a subtraction, which causes a loss of significance (although theoretically they should cancel). x[i-1] is larger than x[i-2], so the error grows each time.

The result will differ depending on the implementation of floating point arithmetic in each compiler and on other hardware choices.