**Computational Physics Exercise 1**

# quadratic solver

It is useful to be able solve quadratic equations. I have written a program which prompts the user to enter a quadratic equation and then calculates the roots using the quadratic formula. The program has various checks and will return various numbers of real or imaginary solutions depending on the inputs, see fig 1.

|  |  |  |  |
| --- | --- | --- | --- |
| coefficients entered | | | Program result |
| a | b | c |
| 1.67 | -12.9 | 2.335 | There are two distinct solutions: x=7.539090 and x=0.185461 |
| 2 | 4 | 2 | There is only one solution: x= -1.000000 |
| 7.6 | 3 | 45 | There is no real solution.  Imaginary solutions are -0.197368 + 2.45304i and -0.197368 – 2.425304i |
| 4.5 | K | -9 | One of the numbers you entered was not read in correctly.  Possibly you did not enter a number? |
| 0 | 6.2 | 3.4 | Equation is linear, not quadratic.  The line will intersect the x-axis at x= -0.548387 |
| 0 | 0 | 5.9 | The equation you entered is a horizontal line.  There are no solutions. |
|  |  |  |  |

*Fig 1: the outputs of program one for various inputs*

Plugging these values back into a calculator I find that the results here solve the quadratic to within 10-6, which certainly seems adequate.

I reduced the accuracy of my program by changing all the variables of type double to floats and got it to solve the equation *x2+40x+3=0*. It found roots of -0.075141 and -39.924858. These roots satisfy the equation to within 10-5. This is surprisingly only slightly less accurate than before.

This program successfully solves an equation of up to order 2 with a satisfactory degree of accuracy. To improve it I would like to add in a graph plotter similar to the one used in problem 3, in order to help visualize the roots. Ideally it would create a 3D graph with x and y being the real and imaginary parts of the input and z being either the absolute value or the real value of the result. Hence a surface would be produced which would intersect the xy plane at the roots found by the rest of the program.

# series expansion of an expression

All computers calculate trigonometric functions by using series expansions. I have written a program which calculates sin(x) from its taylor expansion:

Naturally we cannot actually calculate an infinite sum so we need to investigate how many terms are needed for a reasonable approximation. To do this, my program first outputs data to a text file, this can then be plotted to see how many terms are needed to produce a good fit. This is the resulting graph, plotted by excel:

*Fig2: a graph showing the various approximations of sine achieved when taking a certain number of terms of the Taylor expansion.*

As we can see N=0 produces a poor approximation which appears to only be valid for x close to zero. As we use higher values the approximation improves with N=6 allowing accurate values of sin(x) to be calculated for the entire range -4 to 4. However we then note that N=8 produces a worse approximation than N=6, with the approximation getting progressively worse as N increases further (N=18 did not even produce numbers, only #IND00 errors). Mathematically this is bizarre. However it makes sense when you consider that (2\*8+1)! is approximately 4\*1014; this is simply too big for my factorial function to handle. The factorial function is of type ‘long unsigned int’, which means it can return values of up to about 109. For N=6, it returns (2\*6+1)! =1932053504, which is wrong; the correct value is closer to 6X109. Nevertheless, it is just about the correct order which means that N=6 is a good-ish approximation. We can clearly see that for values bigger than N=6 the incorrect return value of the factorial function will cause larger and larger discrepancies in the calculation of sin(x).

So we have discovered that about 6 terms of the taylor expansion gives as good approximation as is possible. Next we use our taylor expansion to calculate the sine of some well known values and observe how accurate it is, shown in fig 3.

|  |  |
| --- | --- |
| input | Calculated value of sin(x) |
| 0.000000 | 0.000000 |
| 1.570796 | 1.000000 |
| -0.523598 | -0.499999 |

In general then it seems that N=6 calculates sin(x) to a good degree of accuracy for -pi<x<pi.

To improve the program I would like to make it so it doesn’t break if you enter a letter instead of a number by accident.

Fig 3: the output of program 2 for various inputs

# root finding of a polynomial

Quadratic equations are easy to solve. However, often higher order polynomials will need to be solved which cannot be done with a simple formula. I have written a program which finds a root of a polynomial by using Newton-Raphson iteration. In order to determine what value to start the iteration from, the program first outputs data to a text file which can then be plotted in order to get a rough idea of where the roots are. Here is the graph produced for the given equation:

Fig 4: a graph of f(x), using data produced by program 3

Now that we have a rough idea of where the roots are we can perform Newton-Raphson iteration. Fig 4 shows which root is found for various starting positions:

|  |  |
| --- | --- |
| Starting value | Root found |
| start<-2.9 | -3.823050 |
| -2.8<start<-2.5 | 2.930454 |
| -2.4<start<-0.3 | -1.000000 |
| -0.2<start<-0.1 | -3.823050 |
| 0<start< 0.1 | 2.930454 |
| 0.2<start<1.8 | 0.892596 |
| 1.9 | 2.930454 |
| 2 | -1.000000 |
| 2.1 | -3.823050 |
| 2.2<start | 2.930454 |

We can see that the root found is often surprising. Particularly around x=2, Newton-Raphson converges to some very surprising values. None the less 4 separate roots have been successfully found. The number of iterations required varies within the given ranges in fig 5. Except for extremely large or small x, generally less than 10 iterations are required.

It is quite laborious to have to enter a start value each time. I would like to improve the program by making the program automatically investigate many start points and then collate the data into roots. It could also output information about which ranges of start values con verge to which root, perhaps with another graph.

Fig 5: The roots found by Newton-Raphson Iteration for a given starting value.