Homework 01

Ruiqi Feng

September 2022

1 Quadratic Functions

1.1 Description

In order to solve quadratic equations numerically, a Fortran program is created. By taking the input coefficients and inserting them into the root finding equation

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \tag{1}$$

$$x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} \tag{2}$$

my program prints out these solutions. When $b^2 - 4ac < 0$, or the solutions are complex, they are also given.

If $\frac{4ac}{b^2} < 10^{-3}$, the equation is processed in another way to avoid potential underflow in the numerator of Eq.1 when $b > 0$ or Eq.2 when $b < 0$, as is shown in the pseudocode.

1.2 Inputs and Outputs

Testcase	Inputs			Outputs	
No.	a	b	c	x_1	x_2
1	1	2	-1	0.414213538	-2.41421366
2	1.2	3.4	5.6	-1.41666663 + 1.63086545i	-1.41666663-1.63086545i
3	1	2	3	-1.00000000+1.41421354i	-1.00000000-1.41421354i
4	1E-6	1	1	-1.00000107	-999999.000
5	1	1	1E-6	-1.00000102E-06	-0.999998987

Test cases 1, 2 and 3 are standard cases, and the results are accurate. In test cases 4 and 5, $\frac{4ac}{b^2}$ is very small and therefore direct application of Eq.1 and Eq.2 will introduce a large error. Because in my program this case is treated carefully as is described in Sec.1.1, the solutions in test cases 4 and 5 are quite accurate, which can be seen from the table above.

1.3 Pseudocode

Algorithm 1 Solution to equation $ax^2 + bx + c = 0$

$\Delta \leftarrow b^2 - 4ac$
if $b \neq 0$ & $|\frac{4ac}{b^2}| < 10^{-3}$ **then**
  if $b > 0$ **then**
    $x_1 \leftarrow \frac{2c}{-b-\sqrt{\Delta}}$
    $x_2 \leftarrow \frac{-b-\sqrt{\Delta}}{2a}$
  else
    $x_1 \leftarrow \frac{2c}{-b+\sqrt{\Delta}}$
    $x_2 \leftarrow \frac{-b+\sqrt{\Delta}}{2a}$
  end if
else
  if $\Delta >= 0$ **then**
    $x_1 \leftarrow \frac{-b+\sqrt{\Delta}}{2a}$
    $x_2 \leftarrow \frac{-b-\sqrt{\Delta}}{2a}$
  else
    $x_1 \leftarrow \frac{-b}{2a} + \frac{\sqrt{-\Delta}}{2a}i$
    $x_s \leftarrow \frac{-b}{2a} - \frac{\sqrt{-\Delta}}{2a}i$
  end if
end if

2 Game of 24

2.1 Description

The problem to be solved is the classical 24 game: when given 4 cards each assigned an integer below 13 (4 playing cards), find a way in which those cards can produce 24 using 4 operations ($+$, $-$, $\times$ and $\div$).

The approach I take contains two steps. There are 2 major types of possible solutions:

$$(((a \diamond_1 b) \diamond_2 c) \diamond_3 d) = 24 \tag{3}$$

or

$$(a \diamond_1 b) \diamond_2 (c \diamond_3 d) = 24 \tag{4}$$

where each $\diamond$ symbol stands for an arbitrary operation these 6 operations: $+$, $-$, $\times$, $\div$, 'be subtracted by' and 'be divided by'. The former type (Eq.3) will be called a 'sequential' solution while the latter one (Eq.4) will be called a 'combined' solution.

My program first tries to find the 'sequential' solutions. Because the inverse of $\diamond x$ is still in the set of $\diamond x$, a solution of the form $(a \diamond b) \diamond (c \diamond d) = 24$ can be transformed into

$$a = (((24(\diamond_3 d)^{-1})(\diamond_2 c)^{-1})(\diamond_1 b)^{-1}) \tag{5}$$

The algorithm I used is just brute force. I iterated the possible b in Eq.5 and then the $\diamond$ on its left, and after that I iterate c and its $\diamond$ and so on. Every time the RHS of Eq.5 is calculated, it is compared with a on the LHS of Eq.5. When they are close enough, the solution is printed and only the first solution found will be printed. Float type is used to store the calculation results because sometimes the only solution involves fractional calculations. For details please look up the pseudocode.

But when the solutions are all in the form of 'combined' solutions and none of them can be converted to the 'sequential' ones, the 'combined' solutions must be taken into account. For example, $24 = (11 - 9) * (13 - 1)$.

Similarly, the 'combined' solutions are searched by iterating all possible combinations of cards and operations. There are at most only 3 different ways to choose the combination of cards because $C_4^2 = 3$, and they are $\{\{a, b\}, \{c, d\}\}$, $\{\{a, c\}, \{b, d\}\}, \{\{a, d\}, \{b, c\}\}$ respectively. In the pseudocode, ξ_1 and ξ_2 are the first pair and η_3, η_4 are the second pair. The order of ξ_1 and ξ_2 does not matter and so do η_1 and η_2.

2.2 Inputs and Outputs

test case 1 Ordinary

Input:

$$1, 2, 3, 4$$

Output:

$$\begin{aligned}
&\text{solution found}\\
&4.00000000 * 3.00000000 = 12.0000000\\
&12.0000000 * 2.00000000 = 24.0000000\\
&24.0000000/1.00000000 = 24
\end{aligned}$$

One correct 'sequential' solution is printed.

test case 2 Fractional Operation 1

Input:

$$3, 3, 8, 8$$

Output:

$$\begin{aligned}
&\text{solution found}\\
&8.00000000/3.00000000 = 2.66666675\\
&3.00000000 - 2.66666675 = 0.333333254\\
&8.00000000/0.333333254 = 24
\end{aligned}$$

This is called by some 'the most difficult 24 game in the world' because the solution involves fractional operations. My program is capable of solving this kind of problem because float numbers are used.

test case 3 Fractional Operation 2

Input:

$$2, 5, 5, 10$$

Output:

solution found

$2.00000000/10.0000000 = 0.200000003$

$5.00000000 - 0.200000003 = 4.80000019$

$4.80000019 * 5.00000000 = 24$

Another example of a recognized hard puzzle using fractional operations.

test case 3 'Combined' Solution

Input:

$$1, 9, 11, 13$$

Output:

solution found

$1.00000000 - 13.0000000 = -12.0000000$

$9.00000000 - 11.0000000 = -2.00000000$

$-2.00000000 * -12.0000000 = 24.0000000$

This is a case where no 'sequential' solution exists, so my program found a 'combined' solution.

2.3 Pseudocode

Algorithm 2 Find Solutions to the Game of 24

Require: the points of cards x_1, x_2, x_3, x_4 are integers

```
cards ← {x_1, x_2, x_3, x_4}
operations ← {+, −, ×, ÷, 'be subtracted by', 'be divided by'}
for all d in cards do
    for all ◇_1 in operations do
        for all c in cards except the chosen d do
            for all ◇_2 in operations do
                for all b in cards except the chosen d and c do
                    for all ◇_3 in operations do
                        result ← (((24 ◇_1 d) ◇_2 c) ◇_3 b)
                        if |result − a| < 10^{−5} then
                            break from all the loops
                        end if
                    end for
                end for
            end for
        end for
    end for
end for
if no 'sequential' solution is found then
    for all 3 possible combinations do
        according to this combination divide cards into 2 pairs
        ξ_1 and ξ_2 ← one pair
        η_1 and η_2 ← the other pair
        for all ◇_1 in operations do
            for all ◇_2 in operations do
                for all ◇_3 in operations do
                    result ← (ξ_1 ◇_1 ξ_2) ◇_3 (η_1 ◇_2 η_2)
                    if |result − 24| < 10^{−5} then
                        break from all loops
                    end if
                end for
            end for
        end for
    end for
end if
print the solution found
```