Newton-Raphson Inverse

February 1, 2020

1 Algorithm

This code computes the inverse of a number using the Newton-Raphson formula:

$$x_{n+1} = x_n(2 - ax_n) \tag{1}$$

2 What to focus on

1. how COBOL implements function
2. how COBOL implements data types
3. COBOL divisions

3 Notes

The java version of the code is straightforward. After checking the input and some obvious cases, the algorithm is implemented using a function. We compare a "vanilla" java version with the java version using BigDecimal. We'll see the difference in a bit.
In COBOL things are a bit more verbose. First off, note the keyword "Division". We have 3 divisions. One with the name of the program (IDENTIFICATION DIVISION), one with the list of data types (DATA DIVISION), and one with the program proper (PROCEDURE DIVISION).
The second aspect is data types. These are part of the DATA DIVISION and are **not** the usual int, float, char, etc. Instead, we use pictures (keyword:

PIC). Pictures explain how the data should appear on screen. For instance, S99, means that there is a sign ("S") followed by a digit ("9") followed by another digit ("9"). Thus, this PIC means that we can represent numbers from -99 to +99. +100, -134, etc. are **not** allowed. Other useful placeholders in PIC are V for the location of the decimal point, and the parenthesis to represent how many times a digit repeats. For instance, 9V9 represents numbers from 0.0 to 9.9 (no sign!). Instead 9(5) is equivalent to 99999.
COBOL has lots of placeholders, but remember that it does not use the "usual" data types. Another thing to notice is the 01 at the beginning of the line. We will discuss it later on. For now just add it. The VALUE keyword, of course, initializes the variable.
The last point of interest is the "function". In COBOL it is often used the term "subroutine". We will explore this further later on. For now, notice that the subroutine starts with its name "NRinv" followed by a full stop ".". No arguments are passed, no types specified, etc. It is used in the code by just calling it. There are other ways of implementing functions. We will see those later on.

4 Translation

How to convert this code into java? At first, you may think that code "NR-reciprocal.java" works fine. But it does not. The problem is the clash between PIC and data types. As we are computing a division, java represents data using floating point math. BUT our cobol code limits the answer to 5 digits in the decimal part. Here is the catch. It is **not** printing only 5 digits, it actually performs **all** the math using only 5 digits. The way around it, consists in using the package BigDecimal. In both java examples I print 6 decimals. In this way you can see that BigDecimal is actually only using 5 decimals for the math.
Another issue is the way precision is handled. By default cobol drops the extra digits, no rounding unless the keyword ROUNDED is used. Truncation is not present in java, so you must implement it. To see the effect of rounding or truncation, compute the inverse of 7 or -7.
Concerning kotlin, if you understand the java version with BigDecimal, it should be straightforward. You should appreciate the much leaner code.