SOEN 6011

$sinh(x)$

Problem - 3

Pseudo-code and Algorithms

Author

Kolimi Dineshkumar Babu

Problem 3 - F3: $sinh(x)$

SOEN 6011 - Summer 2022 **Kolimi Dineshkumar Babu**
Software Engineering Processes **40094976**
Repository address : https://github.com/dineshkumarkolimi/Soen6011

AExponentiation of integers can be achieved using an algorithm by multiplying or dividing 1 by a predetermined number (e). Exponentiation is substantially more difficult when working with non-real numbers because there are two different ways to calculate them. Similar to finding the exponent, finding the root necessitates constantly testing your hypotheses until you find the appropriate nth exponent. Therefore, it is necessary to balance calculating time and precision.

For calculating real exponents, the natural logarithm, $ln\ x$, can be related to the exponential function, e^x. As an alternative to the previous approach, one can convert real exponents whose values are irrational into rational ones, for instance, $e^{\frac{a}{b}}$, where x^a is divided by $\sqrt[b]{x}$ and hence is chosen as it is easier to code.

Based on the aforementioned considerations, the power and square root functions are the subordinate functions needed to calculate $sinh(x)$, and in addition to these, a function to determine the greatest common denominator can help decrease fractions to make them less computationally costly. This article contains the algorithms for all of the aforementioned functions, as well as an absolute value function that was designed to be straightforward.

- I have chosen Algorithm 2 for this project in the below listed algorithms.

```
function E_POWER_X(x)
    sum ← 1
    for i ← max_steps to 0 do
        sum ← 1 + x × sum/i
    end for
return sum
end function
function CALCULATESINH(epowerX, epowerMinusX)
return (epowerX − epowerMinusX)/2
```

Algorithm 1 Taylor Series

```
max_steps ← 15
function INTCALCULATION(x)
    e^x ← e_power_x(x)
    e^x ← e_power_x(−x)
    function CALCULATESINH(epowerX, epowerMinusX)
    end function
```

Algorithm 2 Power

```
function POWER(base, exp)
    result ← 1
    for i ← 0 to |exp| do
        if exp > 0 then
            result ← result × base
        else
            result ← result / base
        end if
return result
```

Algorithm 3 Hyperbolic Sine

```
function HYPSINE(input)
    if input = 0 then return 0
    end if
    intPart ← input ÷ 1, fractionalNumerator ← input mod 1    ▷ Separating the integral and
real parts.
    fractionalDenominator ← 1
    while fractionalNumerator > fractionalDenominator do
        fractionalDenominator ← fractionalDenominator × 10
    end while
GCD(fractionalNumerator, fractionalDenominator)
    left ← POWER(e, intPart), right ← POWER(e, −intPart)
    if fractionalNumerator > 0 then
        numPower ← POWER(e, fractionalNumerator)
        leftRoot = ROOT(fractionalDenominator, numPower)
        left ← left × leftRoot
        numCalc ← POWER(e, −numPower)
        rightRoot = ROOT(fractionalDenominator, numCalc)
        right ← right × rightRoot
    end if
return (left−right) / 2
end function=0
```

Bibliography

[1] http://www.mathcentre.ac.uk/resources/workbooks/mathcentre

[2] hyperbolicfunctions.pdf

[3] https://www.analyzemath.com/DomainRange/domain$_{r}ange_{f}unctions.html$